IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Patent Application

Applicant(s):	S. Acharya et al.
Case:	18-18-18
Serial No.:	10/659,757
Filing Date:	September 10, 2003
Group:	2127
Examiner:	Kenneth Tang
Title:	Adaptive Scheduling of Data Delivery in a Central Server

---

DECLARATION OF PRIOR INVENTION UNDER 37 C.F.R. §1.131

We, the undersigned, hereby declare and state as follows:

1. We are the named inventors on the above-referenced U.S. patent application.

2. We conceived an invention that is the subject of one or more claims of the application at least as early as August 10, 1998.

3. We reduced the invention to practice at least as early as December 10, 1998. On or about this date we implemented an embodiment of the invention in software code and tested the software code via simulations.

4. Subsequent to the reduction to practice, we prepared a paper entitled "Scheduling Data Delivery Over Multiple Channels," that described the invention, its implementation in software code, and the simulation results.

5. On or about January 26, 1999, the paper was entered as Technical Memorandum Document No. BL0112370-990126-02TM into the Lucent Technologies technical document filing system. A copy of the Technical Memorandum Document, including the paper referred to above, is attached hereto as Exhibit 1.

6. All statements made herein of our own knowledge are true, and all statements made on information and belief are believed to be true.

7. We understand that willful false statements and the like are punishable by fine or imprisonment, or both, under 18 U.S.C. §1001, and may jeopardize the validity of the application or any patent issuing thereon.

Date: 9/16/04 SAcharya

Swarup Acharya

Date: ____________ ______________________

Shanmugavelayut Muthukrishnan

Date: 9|20|04 [illegible]

Ganapathy Sundaram

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Patent Application

Applicant(s):	S. Acharya et al.
Case:	18-18-18
Serial No.:	10/659,757
Filing Date:	September 10, 2003
Group:	2127
Examiner:	Kenneth Tang
Title:	Adaptive Scheduling of Data Delivery in a Central Server

DECLARATION OF PRIOR INVENTION UNDER 37 C.F.R. §1.131

We, the undersigned, hereby declare and state as follows:

1. We are the named inventors on the above-referenced U.S. patent application.

2. We conceived an invention that is the subject of one or more claims of the application at least as early as August 10, 1998.

3. We reduced the invention to practice at least as early as December 10, 1998. On or about this date we implemented an embodiment of the invention in software code and tested the software code via simulations.

4. Subsequent to the reduction to practice, we prepared a paper entitled "Scheduling Data Delivery Over Multiple Channels," that described the invention, its implementation in software code, and the simulation results.

5. On or about January 26, 1999, the paper was entered as Technical Memorandum Document No. BL0112370-990126-02TM into the Lucent Technologies technical document filing system. A copy of the Technical Memorandum Document, including the paper referred to above, is attached hereto as Exhibit 1.

6. All statements made herein of our own knowledge are true, and all statements made on information and belief are believed to be true.

7. We understand that willful false statements and the like are punishable by fine or imprisonment, or both, under 18 U.S.C. §1001, and may jeopardize the validity of the application or any patent issuing thereon.

Date: ____________ ______________________________

Swarup Acharya

Date: 9/28/04 [illegible]

Shanmugavelayut Muthukrishnan

Date: ____________ ______________________________

Ganapathy Sundaram

Document Number(s): ITD-99-36192N; BL0112370-990126-02TM
Title: Scheduling Data Delivery Over Multiple Channels
Author(s): Acharya, S; Muthukrishnan, S; Sundaram, G
Date: 01/26/1999
Pages: 18
Abstract: The boom in the internet and the rise of new network technologies have focused attention on designing faster and more efficient data networks. A key component of the data network is the data server. Data servers are the engines that store and feed content to diverse clients over the network media. Data servers can take many forms, such as infostations, wireless gateways, web servers, or, specialized servers such as traffic and weather information servers. In this paper, we study the issues driving the performance of data servers and explore the scheduling problems that arise in their data delivery. In particular, we focus on a wireless data server having multiple downlink channels at its disposal. This multi-channel system raises a number of novel scheduling issues and we identify and categorize these problems. We highlight different QoS metrics suited to this environment. Besides, the traditional response time metric, we also incorporate the recently proposed stretch metric into our study. Finally, we use a detailed simulation study to explore the various tradeoffs in scheduling for multiple channels. Our results show that optimizing for the maximum/average stretch metric is effective across many different scheduling scenarios.
Key Words: scheduling; stretch; wireless systems; multi-channel systems
Authoring organization: BL0112370
Mercury categories: CMM; MKT; WCL
File case: 39394
Charge case: 311403-3708
Proprietary class: Lucent Technology Proprietary

Lucent Technologies
Bell Labs Innovations

Document Cover Sheet for Technical Memorandum

Title: Scheduling Data Delivery Over Multiple Channels

Authors	**Electronic Address**	**Location**	**Phone**	**Company**
S. Acharya	swarup@bell-labs.com	MH 2B-306	908-582-4321	
S. Muthukrishnan	muthu@research.att.com			AT&T Labs
Ganapathy S. Sundaram	ganeshs@bell-labs.com			Bell-labs

Document No.	**Filing Case No.**	**Work Project No.**
BL0112370-990126[illegible]-02TM	39394	311403-3708

Keywords: Scheduling, Stretch, Wireless Systems, and Multi-channel systems

Abstract

The boom in the internet and the rise of new network technologies have focused attention on designing faster and more efficient data networks. A key component of the data network is the data server. Data servers are the engines that store and feed content to diverse clients over the network media. Data servers can take many forms, such as infostations, wireless gateways, web servers, or, specialized servers such as traffic and weather information servers.

In this paper, we study the issues driving the performance of data servers and explore the scheduling problems that arise in their data delivery. In particular, we focus on a wireless data server having multiple downlink channels at its disposal. This multi-channel system raises a number of novel scheduling issues and we identify and categorize these problems. We highlight different QoS metrics suited to this environment. Besides, the traditional response time metric, we also incorporate the recently proposed stretch metric into our study. Finally, we use a detailed simulation study to explore the various tradeoffs in scheduling for multiple channels. Our results show that optimizing for the maximum/average stretch metric is effective across many different scheduling scenarios.

```
LUCENT TECHNOLOGIES POST
ITDS
ROOM 1A-127
101 CRAWFORDS CORNER RD
HOLMDEL, NJ 07733
```

Order Number: ITD-99-36192N

Pages of text: 15 Other pages: 0 Total: 15
No. Figs.:01 No. Tables: 08 No. Refs.: 19

ily
:api

Scheduling Data Delivery Over Multiple Channels
(Acharya, S)

Initial Distribution Specifications **BL0112370-99012602TM** *(page ii of ii)*

"Cover Sheet Only"	Cover Sheet Only
MTS 1123	Directors
	DPH 112
	MTS 112 (Excluding 1123)
	A. Netravali
	R. Sethi
	A. Silberschatz
	I. Saniee

Future Lucent Distribution by ITDS **Release to any Lucent employee (excluding contract employees)**

Author Signatures

[illegible]		
S. Acharya	S. Muthukrisnan	G. S. Sundaram

Dept. Head Signatures

[illegible]
Henry F. Korth

For use by Recipient of Cover Sheet:
Computing network users may order copies via the *library-1* command; for information, type *man library* after the UNIX system prompt.

Internal Technical Document Service:

( ) AK 2H-28	( ) IH 7M-103	( ) DR 2F-19	( ) NW-ITDS
( ) ALC 1B-102	( ) MV 3L-19	( ) INH 1C-114	( ) PR 5-2120
( ) CB 30-2011	( ) WH 3E-204	( ) IW 2Z-156	
( ) HO 4F-112		( ) MT 3B-117	

Return this sheet to any ITDS location.

Scheduling data delivery over multiple channels

Swarup Acharya
Bell Laboratories
Murray Hill, NJ
acharya@bell-labs.com

S. Muthukrishnan
AT&T Laboratories
Florham Park, NJ
muthu@research.att.com

Ganapathy S. Sundaram
Bell Laboratories
Whipanny, NJ
ganeshs@bell-labs.com

Abstract

The boom in the internet and the rise of new network technologies have focused attention on designing faster and more efficient data networks. A key component of the data network is the data server. Data servers are the engines that store and feed content to diverse clients over the network media. Data servers can take many forms, such as infostations, wireless gateways, web servers, or, specialized servers such as traffic and weather information servers.

In this paper, we study the issues driving the performance of data servers and explore the scheduling problems that arise in their data delivery. In particular, we focus on a wireless data server having multiple downlink channels at its disposal. This multi-channel system raises a number of novel scheduling issues and we identify and categorize these problems. We highlight different QoS metrics suited to this environment. Besides, the traditional response time metric, we also incorporate the recently proposed stretch metric into our study. Finally, we use a detailed simulation study to explore the various tradeoffs in scheduling for multiple channels. Our results show that optimizing for the maximum/average stretch metric is effective across many different scheduling scenarios.

1 Introduction

There are three main components of a data network: the network media (wireline, wireless, satellite or cable-based) with associated protocols, content sources (web pages, audio/video, message file systems etc.) with associated data formats, and data servers (web servers, information kiosks, broadcast servers, etc.) with associated scheduling strategies. With the boom in use of the internet, satellite, wireless and cable-based technologies, and application-level trend towards integrating various services with data, there is an increased need to examine the design and performance of each of these components. There has been focused research on designing faster networks with robust network protocols, better context sources with new data formats, and to some extent, responsive data servers with efficient scheduling strategies. In this paper, we focus on the performance of the data servers: in wireless networks, data servers have many different forms. We identify and abstract the range of scheduling problems that arise during data delivery in wireless data servers. We also provide a suite of algorithmic solutions to these scheduling problems suitable for the different forms of a wireless data server, and perform an experimental study of the tradeoffs involved.

Data servers are the engines that drive the web revolution by feeding content over the underlying network media. As such, data servers form a significant part of the traditional wireline networks, and scheduling data delivery in such traditional servers is a recognized problem [CHBM97, VH96]. With emerging wireless (radio-based, or satellite-based) support for data networks, there is a scope for abundance of data clients (palm held computers, smart devices etc) and data servers (portable kiosks, infostations etc.). Data servers in wireless networks thus take on different forms: traditional "store-and-feed" servers that store the database of documents locally, or the "retrieve-and-feed" servers that retrieve necessary documents from remote mother databases whenever needed, or the "ad hoc" servers that build up their database over time when deployed, etc. Wireless data servers also supply clients with a wide range of capabilities in storage space, power consumption, and computational power. Scheduling data delivery in wireless data servers is thus more varied than wireline data servers, and of great relevance in emerging global data networks.

As a concrete example of a wireless data server, let us consider the concept of an *infostation* as developed at Rutgers University [FI96, GBMY97, Nim99].

In one of its embodiments, we can visualize data servers at isolated spots (much like gas stations), say near toll booths, connected to the internet or a central database. Users on motor traffic may request documents (email, files, music etc.) in advance through a low bandwidth data connection (e.g., cellular phone. These are prefetched at a suitable infostation; when the user passes through the cell commanded by this infostation, the documents are delivered to the user via a high speed wireless link. It seems natural to employ many channels to deliver data: this helps in offering differentiated services, segregating the user traffic into manageable queues, or using different channels for the push traffic vs the pull traffic, etc. The concept of an infostation is versatile, and many different embodiments of a wireless data server may be derived from it. [GBMY97] provides more details. An observation here is that the "size" of a request (the number of bytes in the document or the estimated time for a delivery) is fairly accurately known before the time of delivery; this may be valuable for planning delivery schedules.

Another concrete example of a wireless data server is a base station on the wireless internet. A base station is a host connected to the wireline internet and all mobile units communicate with the internet via a base station (for a typical architecture, see [Wav98]). Thus a base station sees both uplink as well as downlink traffic. It is most naturally seen as a data server on the downlink. This scenario differs from the infostation depicted above because users directly connected to the internet differ from users at an infostation in the applications they run, in QoS requirements, and in their network protocols.

Other diverse cases of wireless data servers exist, and many more such applications are likely to emerge. Some examples of possible variations in data delivery mechanisms in wireless data servers are: multiple or single channel delivery (depending on the availability of a control channel and the ability to tune rapidly), batched or online delivery, single/multiple simultaneous outstanding requests per client, relative computational power of the centralized server vs the "distributed server" over multiple channels, preemption vs non-preemption (depends on the overhead of dis-(re-)connect protocols and the available time window), etc. Given this diversity, there is no single data delivery scheduling problem that captures all the costs and QoS requirements accurately; rather, there are many different scheduling problems each best suitable for a particular scenario.

Our contributions are as follows. We isolate and identify different categories of scheduling problems that arise in data delivery by wireless data servers; we also discuss suitable metrics to optimize while scheduling the delivery. For some of these problems, known results from the area of Scheduling Theory apply; for some others, we develop new algorithms with *provable* performance guarantees; still others, we leave them open. We also present a suite of heuristic scheduling algorithms; some of these are classical, while others appear here for the first time. We experiment with these suite of algorithms to quantify different tradeoffs in the performance under different modes of data delivery. As expected, none of our algorithms emerge as the clear winner in all the different modes. However, attempting to minimize the maximum/average stretch of a user response (defined in Section 2) proves to be a effective guideline across the different categories. This message emerged in two recent papers, namely, [AM98], and [CHBP98]. In [AM98], the authors explicitly used the guideline of optimizing the maximum stretch. However, their results held only in the broadcast delivery case where there are overlapping requests for common documents. Furthermore, their experiments were for the single channel case. In contrast, our message emerges even in the multiple channel case, and

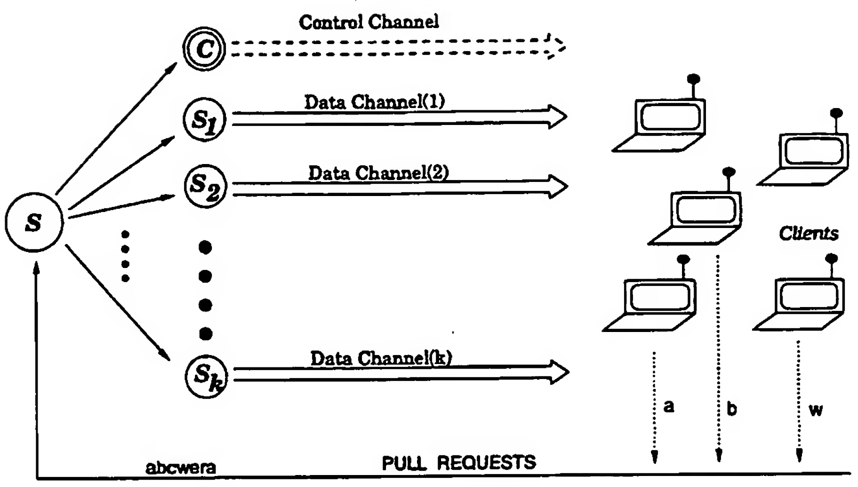

Figure 1: System Model

for the unicast delivery model. In [CHBP98], the authors considered data delivery on conventional data servers using the SRPT algorithm (defined in Section 4) for multiple processors ("channels"); it follows from the recent theoretical result in [GMRS99] that this is tantamount to implicitly using the average stretch as a guideline for scheduling delivery.

Map. The rest of this paper is as follows. In Section 2, we describe the system model of a multi-channel environment and the various scheduling problems that arise therein. In Section 3, we present algorithmic results with provable performance bounds. Then in Section 4, we describe the algorithms that we evaluate experimentally. In Section 5, we present our simulation results. Finally, we present our conclusions.

2 Model and Problems

Model. We use the following basic model of a data server. There is a *centralized server* S which stores the documents. There are *channel servers* $S_1, \ldots, S_k$ where S_i serves the channel C_i; all channels are disjoint. In addition, there is a *control channel* C. *Client appliances* request documents at S. All requests are served through one of the S_i's. At any time, any client appliance may receive data on any one of the S_i's; this information is notified to the client appliance using the control channel C.

This basic model applies to many different data service environments. For example, the centralized server may be a traditional web server with the documents residing in its disk storage, or a infostation server that prefetches the documents requested by the clients in its cell at any time, or the base station in a data network that feeds hosts connected by wireless links, etc. Similarly, channel servers may be processors that can perform powerful computations (including scheduling) on their own such as in a bank of infostation servers, one per channel, or merely signal processors that modulate the data to appropriate channels such as in many proposed wireless base stations.

The basic model admits many variations depending upon the particular data service environment; we look at three categories of these variants. Category I of the variants is as follows.

1. *Centralized vs local scheduling.* This depends on the relative computational powers of S vs S_i's. If S_i's are computationally slim, scheduling has to be done at S to assign the channels to requests as well as the schedule of requests within a channel. On the other hand, if S_i's are sufficiently powerful, it would suffice for S to

merely perform the assignment of the requests to the channels, and let S_i's schedule their assigned requests independently.

2. *Preemptive/Non-preemptive.* Preemption is the flexibility to interrupt the service of a document and resume it latter while having serviced other request(s) in the interim. With non-preemption, requests which are being serviced are processed without interruption until their completion. While preemption leads to better resource utilization and performance metrics, it has an overhead at the servers as well as the client appliances of the increased complexity of control logic and increased buffer space.

3. *Migratory/Non-migratory.* In cases where preemption is allowed, there is the added issue of whether a single request may be serviced on different channels at different times, that is, requests may be migrated from one channel to another when preempted. Migration improves the performance but has the overhead of increased complexity of managing the control channel, as well as switching the client appliance between the channels.

Category I comprises the core variants of the basic model. We will study all Category I variations both theoretically and experimentally. We also identify Category II of variations of the basic model; these we study only theoretically. Category II of variants is as follows.

1. *Single/Multiple simultaneous requests per client appliance.* A system may allow each client appliance to make several requests simultaneously or to make further requests when when there are still outstanding requests by the client. In this case, at most one of the documents requested by a single client may be served on any of the channels at any given time. On the other hand, the system may force the client appliances to make requests in a sequence, each request following another only after the complete service of the previous request. This may happen if the system batches several temporally close document requests of a client into a single session and enforces the clients to have at most one session in progress.

2. *Heterogeneous/Homogeneous Channels.* Channels may be *heterogeneous*, that is, each channel may have a different bandwidth, and therefore different capacity for data; otherwise channels are *homogeneous*. This may also happen when each server S_i controls a bank of several homogeneous channels, and more generally, there is a hierarchy of servers atop the channels.

In addition, we have many other variants which we list as Category III. For example, the client appliances may each be able to access only a subset of the channels, or be able to listen to a set of channels simultaneously etc. The number of available channels may vary over time if the channels are used for other purposes besides delivering documents. There may also be sophisticated dependency in the retrieval order of documents per client appliance. Category III variants are somewhat peripheral in modeling data servers. Some of these variants are easy to solve by modifying our techniques here, while we leave the others open.

Remark 1. Our model and its variations study the data delivery problems at the level of "requests for documents" rather than at that of "request for packets"; thus, the focus is on the application layer rather than the transport layer.

Remark 2. We do not consider engineering issues at the system level, but rather focus on the higher level issues of scheduling data delivery. For example, we do not consider the co-channel interference, (there may be some additional benefit to scheduling requests at non-nearby channels depending on the location of users etc). We also do not deal with errors in transmission.

Remark 3. All our focus is on point-to-point/unicast setting. In wireless data servers, broadcasting is viable. There are many nontrivial issues involved in formalizing the data delivery problems in presence of broadcasting over multiple channels, and we do not study that here (For broadcast-based data delivery over single channel, see the recent paper [AM98]). Also, our setup is *pull-based* rather than *push-based* since client appliances explicitly request documents. In a general wireless data server scenario, one may need to balance the broadcast/unicast and push/pull cases. Having many channels will help since a few of the channels may be used for broadcast/push while reserving the rest for the unicast/pull. Our work is relevant for the unicast/pull channels only.

Scheduling Problems. Our goal is to schedule data delivery in our model in order to optimize various quality-of-service guarantees. We assume that when any request i arrives, we know the time it takes to service it, and denote it by s_i. This calls for modeling the channel characteristics, but a simple model is to divide the byte size of the document by the aggregated channel bandwidth.

The standard QoS criterion for a request is the *response time*, namely, $c_i - a_i$ where c_i is the time when a request was fully serviced and a_i is the arrival time of the request. More recently, the *stretch* measure has been suggested as a fairer criterion, where stretch of a job is $\frac{c_i - a_i}{s_i}$ (where s_i is the size of the request) [BCM98, AM98]. Stretch was proposed in the context of heterogeneous workloads, i.e., requests for data of different sizes (thus, different s_i). The response time metric is skewed towards large jobs since jobs with large service times also tend to have large response time. On the other hand, the stretch metric is independent of size making it more fair to all job classes. Since data requests in the emerging data server applications (such as requests on an infostation) would very likely be heterogeneous, stretch is an attractive metric to investigate.

The overall performance criterion for a set of jobs could be the L_∞ norm, namely, $\max_i(c_i - a_i)$ (max response time)[1] or $\max_i \frac{c_i - a_i}{s_i}$ (max-stretch), or the L_1 norm where $\max_i$ is replaced by $\sum_i$. Since L_1 norm for n jobs is equivalent to the average, the L_1 norm for the flow and the stretch are called average-flow and average-stretch respectively. Our overall goal is to minimize these metrics for any given input instance.

There are two basic variants of our problems, namely *offline* or *online*. In the online case, requests arrive over time and the scheduling algorithms do not know the future requests when processing the current load. In the offline case, all the request arrivals are known ahead of time; in this case, the arrival times are also known as the *release times* since no request may be serviced before its "arrival" time. Offline case is of theoretical interest and is mainly useful to quantify the benefit to be accrued from scheduling, and as a benchmark to compare the various heuristics we study here.

Relation to classical scheduling problems. In essence, our data delivery problems may be thought of as scheduling sequential jobs, that is jobs that may only be run on a single machine at a time, on a parallel or a distributed machine with k processors. We are able to borrow many results known in the classical scheduling literature for our problems. However, when some of our variants (ex. ones in Category II) are mapped to this scenario, we obtain problems that were not previously studied; for some such cases, we provide new algorithms (See Section 3). An example of such a problem is when users may have more than one outstanding request no two of which must be serviced at the same time. This would correspond to the parallel machine scheduling problem with side constraints on groups of jobs that cannot be serviced simultaneously; to the best of our knowledge, this variation has not been studied in the scheduling literature.

[1]We use the terms response time and flow interchangeably.

3 Theoretical results

In this section, we focus on algorithms with provable guarantee bounds for our problems.

3.1 Category I variants

First we focus on the single channel case, i.e., $k = 1$. If preemption is allowed, the first and the only known algorithms with provable performance guarantees for max-flow and max-stretch metrics may be found [BCM98]. First-in-First-Out (FIFO) is optimal for max-flow and MAX gives the best known performance for max-stretch (see [BCM98] for the precise claims, and Section 4 for a description of the algorithms). Average-flow is optimized by the Shortest-Remaining-Processing-Time algorithm (SRPT). For the average-stretch metric, no online algorithm is optimal but SRPT has the best proven performance [GMRS99]. If preemption is not allowed, all these measures are not only NP hard to solve exactly (that is, there are no efficient polynomial time algorithms unless $P = NP$, an outcome that still remains open to resolve), but difficult even to approximate (see e.g., [BCM98, KTW96]).

Now we consider the general case of multiple channels, i.e., $k \geq 1$. There are many variations in our models. These metrics are difficult to optimize (or even approximate) in any of the models if preemption is forbidden, so we focus only on preemptive schedules. For optimizing the max-flow, FIFO has the best known performance [BCM98] and it is an online algorithm. For optimizing the max-stretch (and max-flow) offline, there is an efficient approximation algorithm [BCM98] but no online algorithms are known; we implement and test this offline algorithm for benchmark purposes (the algorithm is described below). A few years ago, it was shown that SRPT (defined in Section 4) has the provably best performance for optimizing the average-response time [LR97]. SRPT also has the best known provable performance for optimizing the average-stretch as was shown very recently [GMRS99]. SRPT may migrate jobs between processors. Recent results indicate that migratory and non-migratory schedules are intimately related [KP99], although no results are known for optimizing average flow/stretch if migration is not allowed.

To summarize, many algorithmic results with provable guarantees are known in the scheduling literature for variants in Category I. In this paper, we perform an experimental study with these algorithms as well as other heuristics (described in Section 4). The best offline algorithms are used as a benchmark as described in Section 4.

3.2 Category II variants

We first consider allowing multiple simultaneous requests by client appliances. As mentioned earlier, there are no previously known results for optimizing various metrics in this variation.

Theorem 3.1 *There exists a polynomial time algorithm to minimize the max-stretch or the max-flow for n requests given offline, with absolute error at most ϵ.*

Proof. This claim holds for the special case when simultaneous multiple requests by any client appliance is forbidden. For that special case, this result was previously known (See [Bru95]). Our theorem is obtained by generalizing that proof.

Fix the problem of optimizing the max-stretch; optimizing the max-flow is similar. First consider the decision problem: *given an input instance, can all requests be serviced with maximum stretch at most T?* Since job i must have stretch at most T, it must complete latest by $s_iT + a_i$ which is its deadline, denoted d_i. The problem reduces to determining if all these deadlines can be met. We solve the problem using the graph maximum flow problem as follows.

Order the arrival times and deadlines of the n jobs in the increasing order and let I_i denote the time interval between the ith and $i+1$th time in this order. Note that the number of such intervals is at most $2n = O(n)$. We have a source vertex s and sink vertex t. We create a vertex for each request r and put an edge from s to r with capacity s_r. We also create a vertex for each interval I_i and put an edge from it to t of capacity $k|I_i|$ where $|I_i|$ is the length of the interval I_i; recall that k is the number of channels.

For each interval I_i, we consider all requests r from a given user appliance u such that $I_i \in [a_r, d_r]$; denote this set as J_i^u. Note that the size of each such set is at most n for a given i; the total size of all such sets is at most $2n^2 = O(n^2)$. For each item in each set J_i^u, we create a vertex and put an edge of capacity $|I_i|$ from it to the sink. Also, we put an edge from request r to the unique vertex representing r if $r \in J_i^u$; this edge too has capacity I_i.

That completes the construction of the graph. As mentioned earlier, this construction is a generalization of that for the special case when no user appliance has more than one outstanding request at any time. The key difference in the constructions is that we need the vertices for sets J_i^u which was not needed in previous constructions.

Now we claim that the maximum flow in this graph is $\sum_i s_i$ if and only if all deadlines can be met, or equivalently, the maximum stretch is at most T. (see [Tar83] for the definition of the maximum flow problem on a graph). This claim is not hard to prove, and follows from the argument for the special case in [Bru95] and the observation that in any time interval of size $|I|$, at most $|I|$ time may be spent servicing requests for a single user appliance although there are k channels and $k|I|$ time may be spent in servicing requests in total.

In order to use the decision problem to determine the minimum possible value of the maximum stretch, we can do a binary search just as in the special case [Bru95, BCM98]. If this value of stretch needs arbitrary precision to represent (note that the stretch for a job may be a rational number), we can get an absolute error ϵ by doing a polynomial number of binary searches. These straightforward details are omitted. The entire process takes polynomial time using the algorithms known for the maximum flow problem on graphs [Tar83]. ■

Our result above can be extended to the case when channels have different bandwidth. This requires a related, but somewhat different reduction to the maximum flow problem on a graph. We omit the details here but the same theorem holds.

Many issues remain open of theoretical interest for the variants in Category II: designing efficient online algorithms to optimize the max-flow or max-stretch, offline/online algorithms to optimize the average flow/stretch, etc. Since the variants in Category I already generate several issues to study experimentally, we chose not to perform an experimental study of the issues that arise in Category II of variants with our results above; hence, our results above are of only theoretical interest. The experimental results for the Category II of variants will be summarized in a future version of this paper.

4 Algorithms studied

In this section, we briefly list the algorithms whose performance we study quantitatively. As listed in Section 2, there are three variations to the basic model in Category I, namely, centralized vs. local scheduling, preemptive/non-preemptive, and migratory/non-migratory. While this allows for eight possible combinations, we study only a subset of them here as described below.

Our experiments have shown that non-preemptive algorithms perform extremely poorly (as would be expected for heterogeneous workloads) and thus, we explore only preemptive algorithms in this paper. In local scheduling algorithms, job migration is not allowed since, by definition, jobs are assigned to channels and they do not migrate.

We study migratory centralized algorithms only since they outperform the non-migratory ones.[2].

The algorithms we experimentally study are as follows:

- *Near-optimal algorithms.* We use the algorithm in Section 3 that reduces the problem of optimizing the maximum flow or stretch to a maximum flow problem on graphs (the algorithm is modified so at any time each user may have only one outstanding job, that is $|J_i^u| = 1$). For $k \geq 1$, we denote this as OPT. For the special case of $k = 1$, the algorithm can be simplified as in [BCM98] and as in that paper, we denote that as BASE. We used the publicly available code for computing the maximum flow on a graph based on the Dinitz algorithm [CG95]. This program provides both maximum flow value and per-edge flow values which we used to derive the response times. Both OPT and BASE are computationally expensive, and they are space consuming. So, we ran them only on small input sizes and used them mainly as a benchmark.

- *Centralized scheduling algorithms.* This approach emulates multi-channel scheduling by using the results of single channel scheduling. When a data channel does not have jobs to run, it requests the centralized scheduler for a new job. The scheduler chooses a candidate among all the pending jobs. Since the problem reduces to scheduling a single channel, we studied the performance of a number of single channel algorithms recently proposed in [BCM98, AM98]. In particular, we present numbers for the three best performing algorithms in that paper. We briefly introduce them below; for details the reader is referred to [BCM98, AM98].

  1. BASE: This is the offline algorithm cited above which is nearly optimal. We use it to study how the best single channel algorithms adapt to the multi-channel framework.
  2. MAX: This is an online heuristic to approximate BASE. In [BCM98, AM98], it was shown to perform well even for response time measures even though it was designed to optimize for stretch. Briefly, MAX maintains the current value of *MaxStr* seen so far. When a request arrives, it is assigned a deadline D time units away where D is given by its job size times the current *MaxStr* value. The jobs are processed using Earliest Deadline First. If a job fails to meet its deadline, that raises the current *MaxStr* value; consequently, the deadlines of all pending jobs are re-evaluated based on this new value.
  3. *Shortest Remaining Processing Time*(SRPT): This algorithm chooses the job to run which has the shortest time left to completion. It is known that SRPT the gives the best proven performance for optimizing the average response time as well as the average stretch for both single channel [KSW98, GMRS99] and multi-channel systems [LR97, GMRS99].

- *Local scheduling algorithms.* In these algorithms, the scheduling decision is taken by each channel locally and independently of each other. When a job arrives in the system, the centralized *dispatcher* merely forwards the job to one of the channels which schedules it locally. Consequently, there are two issues that need to be dealt with here. First, what scheduling algorithm does each channel employ. We studied the two online algorithms from the previous category, namely, SRPT and MAX. For simplicity, we assume all channels follow the same scheduling algorithm though once can conceive of a scenario where each channel is free to choose its own criteria for different QoS requirements. The second and more critical problem is how does the centralized dispatcher choose the channel to assign the job. Note that this is important for performance since once assigned, the job cannot migrate elsewhere. We studied the following three options for dispatching jobs:

[2]However, migration imposes additional overheads which will be described later.

Parameter	**Description**	**Value**
`TotalBW`	Total downlink bandwidth	128Kbits/sec
`NetBW`	Bandwidth per channel	6.4 ⋯ 25.6 Kbits/sec
`NumChannels`	Number of channels	5 ⋯ 20

Table 1: Simulation parameters

1. *Load Balancing* (`LdBal`): In this algorithm, the channel with the smallest load (i.e., fewest pending bytes to transfer) is assigned a new job.
2. *Minimum Flow* (`MinFlow`): In this algorithm, the channel which has the lowest current average-flow is given the new job.
3. *Minimum Stretch* (`MinStr`): In this algorithm, the channel which has the lowest current maximum stretch value is given the new job.

We also studied a few variants of these local scheduling algorithms such as assigning the newly arrived job to the channel where the maximum/average stretch/flow value was changed by the smallest amount upon the addition of the new job, etc. However, the three above have the most interesting performances, so we present only those numbers.

5 Experimental results

In this section, we present performance results for different scheduling algorithms based on a detailed simulation study of a multi-channel environment. We first describe the input traces used in the simulation followed by a description of the simulation model. Finally we present a selection of our experimental observations.

5.1 Input traces

For the simulation, we used input trace data from a real-life WWW server. Our choice of Web data was motivated by two factors. First, web workloads are publicly available. Second, the web data access can be expected to be qualitatively similar to the expected access patterns of next generation data servers especially in the wide variation in server loads and requests of widely varying sizes.

In this paper, we used logs from the IRCACHE project at NLANR [NLA99]. The IRCACHE project maintains a set of caches at each of the U.S Supercomputer Centers and makes their access logs available. The trace used for this experiment consisted of 75,000 accesses for the cache at San Diego for July 7, 1998. About 52,000 distinct objects were accessed with the smallest being 17 bytes and the largest being 20 MB.

5.2 Simulation model and parameter settings

We performed an extensive simulation-based study to explore the multi-channel environment. The simulator was written in C++ using CSIM [Sch86].

The model of the multi-channel environment is similar to what is described in Section 2. It consists of clients making requests to the data server which responds on one or more data channels. We assume that the server has a fixed amount of downlink bandwidth, `TotalBW`, for the data channels divided over `NumChannels` channels giving each channel `NetBW=TotalBW/NumChannels` bandwidth. We do not model the control channel(s) explicitly. The

Max. Stretch	23.2
Avg. Stretch	9.8
Avg. Response Time	12.9 sec

Table 2: Optimal Algorithm Performance

length of a job (i.e., the service time of the request) is determined by dividing the size of the document requested by NetBW.[3]

Table 1 gives the various values for the parameters chosen. For these experiments, we chose a TotalBW of 128 KBits/sec (ISDN bandwidth). In the first set of experiments, we fix the number of channels to 5 while in the second set we vary the number of channel from 5 to 20. While the downlink bandwidth makes a quantitative difference in the performance numbers, the crucial parameter is the ratio of the volume of requests to the bandwidth which determines the relative performance of the scheduling algorithms ("load"). In all our experiments, we choose parameters to ensure suitable values for this ratio of the "relative load" on the server.

We compare the various algorithms on both the flow and stretch metrics. In particular, we study four parameters, namely, average flow time (*AvgResp*), maximum flow time (*MaxResp*), average stretch (*AvgStr*) and maximum stretch (*MaxStr*). Recall that the response (flow) time is the traditional metric used whereas the stretch is a recently proposed metric which is expected to be more "fair" in the presence of workloads with jobs of different sizes ("heterogeneous" workloads) [BCM98, AM98].

5.3 Expt 1: Basic Tradeoffs

In this section, we compare the basic tradeoffs among the three classes of algorithms. We fix the TotalBW and compare their performance for a system with five identical data channels.

5.3.1 Performance of OPT

The algorithm OPT finds near-optimal maximum stretch using the maximum flow problem on graphs. Table 2 shows its performance. The minimum value of *MaxStr* for the given trace found by OPT is 23.2. This will be a benchmark to compare the performance of the online algorithms later.

It is interesting to look at the performance of OPT for the average measures. The *AvgStr* and *AvgResp* values of OPT are significantly worse than what the other algorithms achieve. This is due to at least two reasons. Firstly, the maximum flow formulation was designed to optimize only for *MaxStr*. Since maximum and average are complementary metrics (for stretch), i.e.,, minimizing one necessarily leads to increasing the other, the average values can be expected to be high. The second reason in the way the maximum flow implementation used in the study allocated flows to edges. Our experiments showed that even if there was a scope to schedule a job earlier, the program used a lazy approach to assign jobs to slots delaying the assignment as close to the deadline as possible. Thus, one can potentially improve the average time performance of this maximum flow approach by adapting the code. However, since our goal of using this formulation was to solely derive the maximum flow numbers, we did not adapt the code.

5.3.2 Centralized Scheduling Algorithm Performance

In this section, we study the performance of the centralized online algorithms and evaluate how they compare to OPT.

[3]We use the words "job" and "request" interchangeably. Similarly, we use the words "data items" and "documents" interchangeably.

Algorithm	Stretch		Response Time (sec)	
	Maximum	Average	Maximum	Average
SRPT	34.4	1.1	11878	**3.9**
MAX	**23.98**	1.3	**11775**	5.2
BASE	24.09	**1.08**	11807	4.2

Table 3: Centralized Scheduling Performance

Algorithm	Stretch		Response Time (sec)	
	Maximum	Average	Maximum	Average
LdBal	128.1	4.2	**10085**	11.9
MinFlow	120.6	**1.2**	12463	**4.8**
MinStr	**78.7**	1.3	12019	5.2

Table 4: Local Scheduling Performance, Per-Channel Sched: MAX

Table 3 shows the performance of these algorithms. The numbers in bold represent the algorithm which has the least value. We first compare the stretch numbers. As can be seen, both MAX and BASE have *MaxStr* numbers only slightly above the optimal value of 23.2. This is particularly attractive for MAX since, unlike the BASE and OPT, it is an online algorithm — it is very efficient and has low overhead. An interesting observation is that MAX does better than BASE even though BASE is known to nearly optimize maximum stretch for the single channel case. This implies that single channel solutions cannot be applied directly to the multi-channel case and expected to work well. SRPT has the worst performance for the max. stretch metric, being about 50% higher than the other two. All algorithms have similar average stretch value.

In terms of response time, SRPT has the best provable performance for average response time and the numbers reflect this. MAX is only slightly higher than this optimal value and given its good stretch numbers, it makes a consistent algorithm overall. All algorithms have nearly the same maximum response time with MAX having the lowest value.

5.3.3 Performance of Local Scheduling Algorithms

In this section, we consider the performance of the local scheduling algorithms described in Section 4. Tables 4 and 5 show the numbers for the three dispatch heuristics, LdBal, MinFlow and MinStr for MAX and SRPT respectively. Consider the relative performance of the three heuristics. It shows that load balancing, which is often used as a dispatching heuristic, is by far the worst. This is because once a channel has been assigned a large job, no new jobs are assigned to it; thus, the large job is serviced uninterrupted. In effect, load balancing creates fewer preemptions. Consequently, effective downlink bandwidth of the system is lower leading to poor performance in all categories

Algorithm	Stretch		Response Time (sec)	
	Maximum	Average	Maximum	Average
LdBal	259.4	4.0	**10179**	12.8
MinFlow	305.2	**1.3**	11702	**4.6**
MinStr	**112.1**	**1.3**	12593	4.9

Table 5: Local Scheduling Performance, Per-Channel Sched: SRPT

Number of Unfinished Jobs	MAX	MinFlow	MinStr
Average	41.8	8.1	8.9
Maximum	113.0	22	24

Table 6: Unprocessed Jobs in the System

Channels used per Job	SRPT	BASE	MAX
Average	2.52	2.71	2.53
Maximum	637	907	911

Table 7: Number of channels used per job

except maximum response time (since the large jobs get preference on bandwidth).

The behavior of `MinStr` and `MinFlow` are as expected. While `MinStr` has only slightly higher response time numbers compared to `MinFlow`, the latter has significantly worse *MaxStr* value. Consequently, `MinStr` has a more consistent performance overall.

In comparing the two scheduling algorithms used, SRPT and MAX, the former leads to lower response time numbers while the latter generates better stretch numbers. Consequently, a reasonable compromise which attempts to get the best of both worlds is to dispatch using stretch (`MinStr`) and then, schedule locally using SRPT.

5.3.4 Centralized vs. Local Scheduling Tradeoffs

We now compare the tradeoffs involved in doing centralized versus local scheduling. From the sole viewpoint of optimizing the metrics under consideration, centralized scheduling will always outperform localized scheduling for most metrics. Intuitively, this is because centralized scheduling commits jobs to channels lazily on demand much like a single queue for multiple bank tellers. This guards against sudden variation in load or the nature of the requests in the future.

The quantitative results in the last two sections bear this out. The only metric where a local algorithm has the best numbers is maximum response time (by `LdBal`). The maximum response time is usually due to the largest job. As explained earlier, `LdBal` minimizes the maximum response time by greedily allocated large jobs to a channel and never preempting them. On the other hand, when the large job is part of a single queue, as in centralized scheduling, it typically has to wait its turn while the smaller jobs are being processed. Consequently, solely from a performance viewpoint, centralized scheduling would be the choice for all metrics except possibly maximum response time.

However, centralized scheduling has much higher overhead in general. For example, on the average, MAX had 41.8 jobs unfinished in the system whereas `MinFlow` and `MinStr` has 8.1 and 8.9 jobs per channel respectively (Table 6). The table also presents the maximum number of unprocessed jobs at any point during the simulation run and shows a much higher maximum for MAX. Since the cost of every scheduling decision is directly correlated with the number of waiting jobs, the computational complexity is much higher for the centralized algorithms. Consequently, for the centralized case, the resource requirements for the data server would be significantly higher.

The centralized approach imposes additional overhead on the clients due to migration of jobs. Table 7 shows the average and maximum number of channels used to deliver a job for each of the three centralized algorithms. While migration leads to improved performance, it can cause tremendous signaling overhead at the client since it has to continually switch channels. Besides, there may be physical and radio technology limitations which prohibit frequent or any switching at all. In their current form, none of the centralized algorithms provide any guarantees as to when a

	AvgResp		*MaxStr*	
NumChannels	SRPT	MAX	SRPT	MAX
5	3.9	5.1	34.3	23.9
10	4.8	5.6	17.5	12.6
20	6.3	6.8	9.1	6.7

Table 8: Influence of Number of Channels of Performance

preempted job would be continued again. Consequently, this precludes a mobile client into going into *doze mode* and forces it to continually monitor the control channel.

Finally, the local scheduling algorithms also have their share of overhead. Since the key step in these algorithms is dispatching a job to a channel, for it to be effective, the dispatcher has to be fully aware of the state of each channel. Consequently, these algorithms require continual flow of information, such as load, number of jobs, current values of different metrics etc., from the different channels to the dispatcher.

5.4 Expt 2: Impact of the number of channels

In the last experiment, we studied the performance of the different algorithms for a system with a fixed number of channels. In this experiment, we vary the number of channels (NumChannels) keeping the total bandwidth (TotalBW) fixed. In particular, we experimented with three values of NumChannels, namely, 5, 10 and 20 channels. Table 8 shows the average response time and the maximum stretch numbers for two centralized algorithms, MAX and SRPT.

The table shows a very interesting behavior. Increasing the numbers of channels hurts the response time performance but improves the maximum stretch. This can be explained as follows. Increasing the number of channels keeping the total downlink bandwidth fixed, reduces the bandwidth per channel. In effect, as the number of channels increase, more jobs are allowed to progress but the progress made by each job is smaller for a given time unit. Consequently, even if a job has a channel all to itself, it would take twice as long for 10 channels and it would take for 5. In effect, most jobs take longer to complete.

A natural question then to ask would be, when do having multiple channels help? Note that we study only response time, which measures when the last byte of the request is delivered. Another commonly used metric is latency, which measure the time from which the request is made until the time the *first* byte is delivered. Consequently, increasing the number of channels improves the average system latency. Thus, while choosing the appropriate number of channels is a complex problem, one of the key factors influencing it would be the metric being optimized.

Increasing the number of channels improves the *MaxStr* of the system for some of the same reasons. Since there are more channels available, the jobs are allowed to run longer without preemption. Besides, since the channel bandwidth is lower, it increases the service time of the job, allowing for greater scheduling leeway.

6 Conclusions

In this paper, we study the issues influencing the performance of data servers. Data servers are a key component of a data network which store and deliver content to diverse clients on the network. Data servers can take many forms, such as infostations, wireless gateways, web servers or, emerging applications such as traffic and weather information servers.

In this paper, we have explored the performance issues that arise in a wireless data server and focus on a server with multiple downlink data channels. This multi-channel system raises a number of novel scheduling issues and in this paper, we highlight these. In particular, we identify a subset of these problems for which we propose near-optimal offline algorithmic solutions and efficient online heuristics. We also identify QoS metrics suited to this environment. The traditional metric is the response time; we additionally study the performance of various algorithms for stretch, a metric recently shown to be more fair for heterogeneous workloads. Finally, using extensive simulations, we quantitatively study the online heuristics and show that their performance is very close to that of the optimal algorithm. The experiments also demonstrate that using maximum/average stretch as a basis for optimization proves to be an effective guideline.

References

[AM98] S. Acharya and S. Muthukrishnan. Scheduling on-demand broadcasts: New metrics and algorithms. In *Proc. of IEEE/ACM Mobicom*, Dallas, October 1998.

[BCM98] M. Bender, S. Chakrabarti, and S. Muthukrishnan. Flow and stretch metrics for scheduling continuous job streams. In *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 270–279, San Francisco, California, 25–27 January 1998.

[Bru95] P. Brucker. *Scheduling Algorithms*. Springer-Verlag, 1995.

[CG95] B. Cherkassky and A. Goldberg. On implementing push-relabel method for the maximum flow problem. In *Proc. of IPCO*, pages 157–171, 1995.

[CHBM97] M. Crovella, M. Harchol-Balter, and C. Murta. Task assignment in a distributed system: Improving performance by unbalancing load. Technical Report 97-018, Boston University, October 31, 1997.

[CHBP98] M. Crovella, M. Harchol-Balter, and S. Park. The case for SRPT scheduling in web servers. Technical report, Boston University, October 31, 1998.

[FI96] Dick Frenkiel and Tomasz Imielinski. Infostations: The joy of many-time, many-where publications. Technical report, WINLAB, April 1996.

[GBMY97] D. J. Goodman, J. Borras, N.B. Mandayam, and R.D. Yates. Infostations: A new system model for data and messaging services. In *Proc. of IEEE VTC '97, Vol. 2*, pages 969–973, Phoenix, May 1997.

[GMRS99] J. Gehrke, S. Muthukrishnan, R. Rajamohan, and A. Shaheen. Scheduling to minimize average stretch. Technical Report 99-09, Dimacs, 1999.

[KP99] B. Kalyanasundaram and K. Pruhs. Eliminating migration in multiprocessor scheduling. In *Proc. of ACM/SIAM Symp on Discrete Algorithms*, Baltimore, January 1999.

[KSW98] D. Karger, C. Stein, and J. Wein. *Handbook of Algorithms and Theory of Computation*, chapter Scheduling Algorithms. CRC Press, 1998.

[KTW96] H. Kellerer, T. Tautenhahn, and G. J. Woeginger. Approximability and nonapproximability results for minimizing total flow time on a single machine. In *Proc. of 28th ACM Symp on Theory of Computing*, pages 418–426, 1996.

[LR97] S. Leonardi and D. Raz. Approximating total flow time on parallel machines. In *Proc. of 29th ACM Symp on Theory of Computing*, 1997.

[Nim99] NIMBLE Project Home Page. WWW, URL: `http://www.cs.rutgers.edu/dataman/nimble`, January 1999.

[NLA99] NLANR Home Page. WWW, URL: `http://ircache.nlanr.net`, Jan 1999.

[Sch86] H. Schwetman. CSIM: A C-based process oriented simulation language. In *Proceedings of 1986 Winter Simulation Conference*, 1986.

[Tar83] R. Tarjan. *Data Structures and Network Algorithms*. SIAM Monograph Series, 1st edition, 1983.

[VH96] N. Vaidya and S. Hameed. Data broadcast in asymmetric wireless environments. In *First International Workshop on Satellite-based Information Services (WOSBIS)*, November 1996.

[Wav98] Lucent Technologies, WAVELAN Home Page. WWW, URL: `http://www.wavelan.com`, Feb 1998.